

# Package ‘jvcoords’

December 18, 2018

**Type** Package

**Title** Principal Component Analysis (PCA) and Whitening

**Version** 1.0.2

**Date** 2018-12-09

**License** GPL-3

**URL** <https://github.com/seehuhn/jvcoords>

**Description** Provides functions to standardize and whiten data, and to perform Principal Component Analysis (PCA). The main advantage of this package over alternatives like `prcomp()` is, that `jvcoords` makes it easy to convert (additional) data between the original and the transformed coordinates. The package also provides a class `coords`, which can represent affine coordinate transformations. This class forms the basis of the transformations provided by the package, but can also be used independently. The implementation has been optimized to be of comparable speed (and sometimes even faster) than existing alternatives.

**NeedsCompilation** no

**Author** Jochen Voss [aut, cre] (<<https://orcid.org/0000-0002-2323-3814>>)

**Maintainer** Jochen Voss <[voss@seehuhn.de](mailto:voss@seehuhn.de)>

**Repository** CRAN

**Date/Publication** 2018-12-17 23:20:04 UTC

## R topics documented:

<code>jvcoords-package</code> . . . . .	2
<code>coords</code> . . . . .	2
<code>PCA</code> . . . . .	4
<code>standardize</code> . . . . .	5
<code>whiten</code> . . . . .	6

<b>Index</b>	<b>8</b>
--------------	----------

---

jvcoords-package
*Package overview*

---

### Description

The `jvcoords` package provides functions to standardize and whiten data, and an implementation of Principal Component Analysis (PCA). All three transformations are implemented using a common class `coords` which allows to easily convert data from and to the new coordinate systems.

See the documentation for [standardize](#), [whiten](#), and [PCA](#) for information on how to use this package.

### Author(s)

Jochen Voss <[voss@seehuhn.de](mailto:voss@seehuhn.de)>

### See Also

[standardize](#), [whiten](#), [PCA](#), [coords](#)

---

coords
*An S3 class to represent affine coordinate transforms*

---

### Description

Perform affine coordinate transformations.

### Usage

```

coords(p, name = NULL, shift = 0)
appendTrfm(trfm, op = c("diag", "orth"), val)
toCoords(trfm, x)
fromCoords(trfm, y, apply.shift = TRUE)

```

### Arguments

<code>p</code>	The number of variables in the original data.
<code>name</code>	A short name for the coordinate transformation (optional).
<code>shift</code>	A value subtracted from the data as the first step of the coordinate transformation. Usually, this will be the mean of the data (optional).
<code>trfm</code>	An object of class <code>coords</code> .
<code>op</code>	The type of transformation to append.
<code>val</code>	Data for the transformation to append.
<code>x</code>	Data matrix, rows are observations, columns are variables.
<code>y</code>	Transformed data matrix, rows are observations, columns are variables.
<code>apply.shift</code>	Whether to apply the final shift of coordinates. Set this to <code>FALSE</code> in order to only apply the linear part of the transformation.

## Details

The function `coords()` creates a new object representing an affine coordinate transformation. Initially, the object represents a shift by the amount `shift`, mapping  $p$ -dimensional vectors  $x$  to  $x$ -`shift`. The function `appendTrfm()` can then be used to modify the transformation. The optional argument `name`, if set, is used when printing objects of class `coords`.

The function `toCoords()` applies the affine transformation `trfm` to the data  $x$ . The data  $x$  must either be a vector of length `trfm$p`, in which case the result is a vector of length `trfm$q`, or a matrix with `trfm$p` columns, in which case the transformation is applied to each row of the matrix separately.

The function `fromCoords()` implements the inverse transform to `toCoords()`. The output always satisfies `toCoords(trfm, fromCoords(trfm, y)) == y`. If `trfm$p == trfm$q`, *i.e.* if the transformation is bijective, the `fromCoords(trfm, toCoords(trfm, x)) == x` also holds. The argument `apply.shift` can be set to `false` to apply only the linear part of the (inverse) transformation, leaving out the final shift.

The function `appendTrfm()` concatenates `trfm` with an additional, linear transformation and returns the result. The arguments `op` and `val` specify which kind of linear transformation to append. There are two choices for `op`:

- `diag` denotes multiplication with a diagonal matrix: an input vector  $x$  is mapped to the output  $x * \text{val}$ . The scaling factor `val` can either be a vector of length `trfm$q` (for element-wise scaling), or a number.
- `orth` denotes multiplication with an orthogonal matrix. `val` must be a matrix with orthogonal columns (not necessarily square) and `trfm$q` rows. An input vector  $x$  is mapped to the output  $x \%*\% \text{orth}$ .

The new transformation is applied after any other transformations already associated with `trfm`.

## Value

An object of class `coords`, as a list with the following components:

<code>p</code>	the number of variables in the original data set
<code>q</code>	the number of variables in the transformed data set
<code>shift</code>	the affine part of the transformation
<code>name</code>	the name of the transformation
<code>cmds</code>	a representation of the transformation (internal use only)

## Author(s)

Jochen Voss <[voss@seehuhn.de](mailto:voss@seehuhn.de)>

## See Also

[standardize](#), [whiten](#), [PCA](#)

**Examples**

```
pc <- PCA(iris[, 1:4], n.comp = 3)
toCoords(pc, c(5, 3, 4, 1))
fromCoords(pc, c(1, 0, 0))
```

PCA

*Perform Principal Component Analysis (PCA)***Description**

Perform principal components analysis on a data matrix and return the results as an object of class `coords`.

**Usage**

```
PCA(x, n.comp, scale = FALSE, compute.scores = TRUE)
```

**Arguments**

<code>x</code>	A data matrix, rows are observations, columns are variables.
<code>n.comp</code>	How many principal components to compute.
<code>scale</code>	Whether to standardize the columns before doing PCA.
<code>compute.scores</code>	Whether to compute the scores (i.e. <code>x</code> in the new basis).

**Details**

This function performs Principal Component Analysis (PCA) on the data. Variables are always centred before the PCA is performed and, if `scale` is set, the variables will also be rescaled to unit variance.

If `compute.scores` is set to `FALSE`, only the information required for the `toPC()` and `fromPC()` to work is stored in the returned `coords` object; otherwise the scores will be stored in the `$y` field of the `coords` object.

The `PCA()` function is an alternative to the `prcomp()` command from the standard library. The main advantage of `PCA()` is that the `coords` class provides functions to convert between the original basis and the principal component basis.

**Value**

An object of class `coords`, with the following additional components added:

<code>loadings</code>	the loadings, each column is one of the new basis vectors
<code>y</code>	if <code>compute.scores==TRUE</code> , this is <code>x</code> expressed in the new basis
<code>var</code>	the variance of the data along each of the new basis vectors
<code>total.var</code>	the total variance of the data

**Author(s)**

Jochen Voss <voss@seehuhn.de>

**See Also**

[coords](#); alternative implementations: [prcomp](#), [princomp](#)

**Examples**

```
pc <- PCA(iris[, 1:4], scale = TRUE, n.comp = 2)
pc
plot(pc$y, col=iris$Species)
```

---

standardize

*Standardize data*

---

**Description**

Standardize each column of a data matrix and return the results as an object of class `coords`.

**Usage**

```
standardize(x, compute.scores = TRUE)
```

**Arguments**

`x` A data matrix, rows are observations, columns are variables.  
`compute.scores` Whether to compute the scores (i.e. `x` in the new basis).

**Details**

This function standardizes the columns of `x` by subtracting the mean of each column and then dividing by the standard deviation. The transformed data is stored in the `$y` field of the returned `coords` object.

If `compute.scores` is set to `FALSE`, only the information required for the `toCoords()` and `fromCoords()` to work is stored in the returned `coords` object; otherwise the scores (transformed data) will be stored in the `$y` field of the `coords` object.

**Value**

An object of class `coords`, with the following additional components added:

`y` if `compute.scores==TRUE`, this is `x` expressed in the new basis

**Author(s)**

Jochen Voss <voss@seehuhn.de>

**See Also**

[coords](#); alternative implementation [scale](#)

**Examples**

```
w <- standardize(iris[, 1:4])
colMeans(w$y)
apply(w$y, 2, sd)
```

---

whiten

*Whiten data*


---

**Description**

Whiten data and return the results as an object of class `coords`.

**Usage**

```
whiten(x, compute.scores = TRUE)
```

**Arguments**

`x` A data matrix, rows are observations, columns are variables.  
`compute.scores` Whether to compute the scores (i.e. `x` in the new basis).

**Details**

This function whitens the data by finding an affine transformation such that the transformed data has mean 0 and identity covariance matrix.

If `compute.scores` is set to `FALSE`, only the information required for the `toCoords()` and `fromCoords()` to work is stored in the returned `coords` object; otherwise the scores (transformed data) will be stored in the `$y` field of the `coords` object.

**Value**

An object of class `coords`, with the following additional components added:

`loadings` the loadings, each column is one of the new basis vectors  
`y` if `compute.scores==TRUE`, this is `x` expressed in the new basis

**Author(s)**

Jochen Voss <[voss@seehuhn.de](mailto:voss@seehuhn.de)>

**See Also**

[coords](#)

*whiten*

7

### **Examples**

```
w <- whiten(iris[, 1:4])  
colMeans(w$y)  
round(cov(w$y), 3)
```

# Index

- \*Topic **array**
  - [jvcoords-package, 2](#)
- \*Topic **math**
  - [jvcoords-package, 2](#)
- \*Topic **multivariate**
  - [jvcoords-package, 2](#)
- \*Topic **package**
  - [jvcoords-package, 2](#)
- \*Topic
  - [jvcoords-package, 2](#)
- [appendTrfm \(coords\), 2](#)
- [coords, 2, 2, 4–6](#)
- [fromCoords \(coords\), 2](#)
- [jvcoords-package, 2](#)
- [PCA, 2, 3, 4](#)
- [prcomp, 5](#)
- [princomp, 5](#)
- [scale, 6](#)
- [standardize, 2, 3, 5](#)
- [toCoords \(coords\), 2](#)
- [whiten, 2, 3, 6](#)